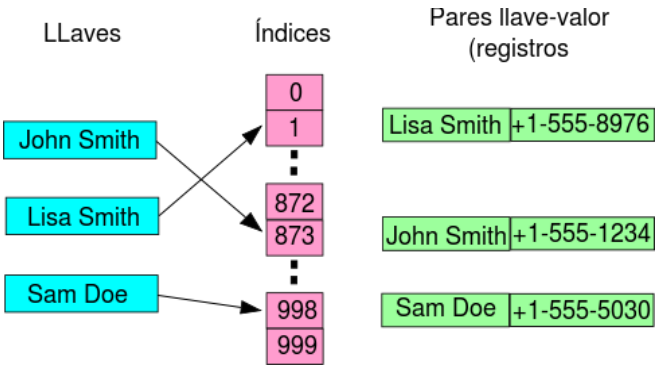
Tablas Hash

Una tabla Hash es un contenedor asociativo (*tipo* ***Diccionario***) que permite un almacenamiento y posterior recuperación eficientes de elementos (*denominados* ***valores***) a partir de otros objetos, llamados **claves o llaves**.

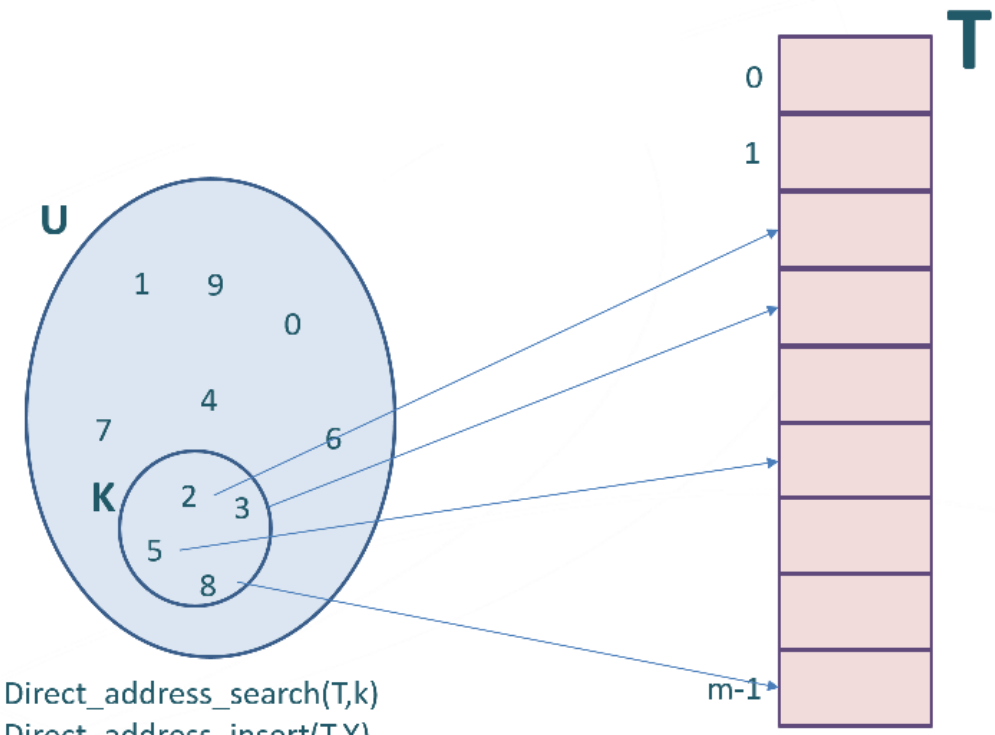
Una tabla hash se puede ver como un conjunto de entradas. Cada una de estas entradas tiene asociada una clave única, y por lo tanto, diferentes entradas de una misma tabla tendrán diferentes claves. Esto implica, que una clave identifica unívocamente a una entrada en una tabla hash. Por otro lado, las entradas de las tablas hash están compuestas por dos componentes:

- La propia clave
- La información que se almacena en dicha entrada.



Ejemplo: Se necesita organizar los periódicos que llegan diariamente de tal forma que se puedan ubicar de forma rápida, entonces se hace de la siguiente forma - se hace una gran caja para guardar todos los periódicos (*una tabla*), y se divide en 31 contenedores (*ahora es una "**hash table**" o tabla fragmentada*), y la clave para guardar los periódicos es el día de publicación (*índice*). Cuando se requiere buscar un periódico se busca por el día que fue publicado y así se sabe en que zócalo (*bucket*) está. Varios periódicos quedarán guardados en el mismo zócalo (*es decir **colisionan** al ser almacenados),* lo que implica buscar en la sub-lista que se guarda en cada zócalo. De esta forma se reduce el tamaño de las búsquedas de **O(n)** a, en el mejor de los casos, **O(1)** y, en el peor, a **O(log(n)).**

Direccionamiento directo (No es una Tabla Hash)

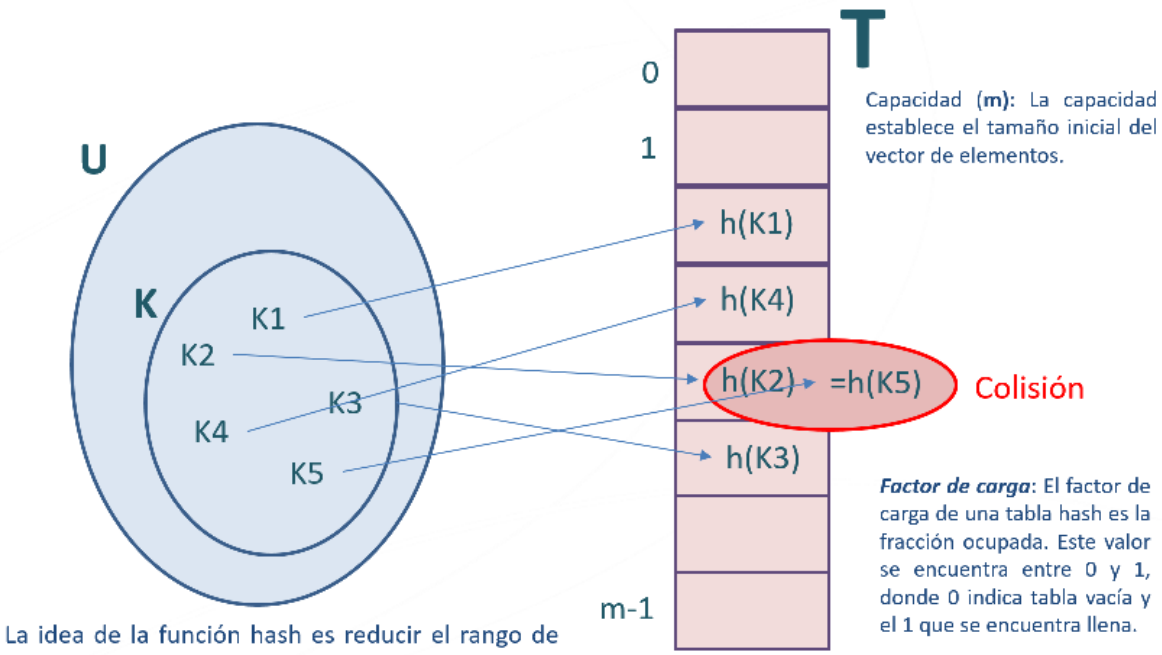


Direct_address_search(T,k)
Direct_address_insert(T,X)
Direct_address_Delete(T,X)

Operaciones de Tiempo Constante **O(1)**

Para representar conjuntos dinámicos, se usa un arreglo o un tabla de direccionamiento directo T[0..m – 1], en la cual cada posición, o slot, corresponde a una llave en el universo de llaves U.

Direccionamiento con Tablas Hash



Objetivo

Definir una función que, dada una clave de búsqueda, determine la posición de almacenamiento del ítem (de datos), similar a la funcionalidad de un "diccionario": teniendo una clave, se accede a un valor (clave, valor) para intentar reducir el tiempo a constante: O(1)

Características

- *No existe un ordenamiento físico de los datos.*
- *Facilita inserción y eliminación rápida de registros por clave.*
- *Encuentra registros por búsqueda asociativa con escasos accesos en promedio.*
- *La teoría de las tablas hash se basa en probabilidades.*

Para implementar una Tabla Hash se necesita

- *Una estructura de acceso (un arreglo).*
- *Una estructura de datos con una clave (datos propiamente dichos).*
- *Una función (función hash).*

Algunas Características

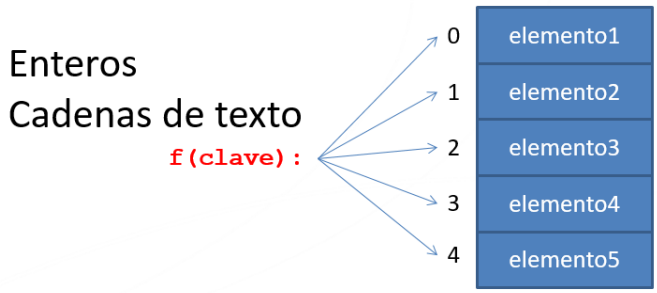
El tiempo medio de recuperación de información es constante, es decir, no depende del tamaño de la tabla ni del número de elementos almacenados en la misma.

Una tabla hash está formada por un **array de entradas**, que será la estructura que almacene la información, y por una **función de dispersión**. Esta función permite asociar el elemento almacenado en una entrada con la **clave** de dicha entrada; por lo tanto, es un algoritmo crítico para el buen funcionamiento de la estructura.

Es frecuente que se produzcan **colisiones**, que se generan cuando para dos elementos de información distintos, la *función de dispersión* les asigna la misma *clave*.

Función Hash

Es una operación que consiste en transformar una clave en una posición dentro de la tabla Hash



Es posible que diferentes claves apunten a la misma posición de la tabla provocando **COLISIONES**

- *Función Hash, Modulo*

hash(llave) = llave mod Max
Donde Max es el tamaño de la tabla

Inconvenientes:
-sólo se usa la parte menos significativa de la llave
-puede producir distribuciones muy poco homogéneas
- cuando alguna de las "terminaciones" de las llaves tiende a
repetirse más que otras

- *Función Hash, troceado en grupos de cifras*

$$\text{hash(llave)} = \left(\sum_{i=0}^{n-1} \left(\frac{llave}{max^i} \bmod Max \right) \right) \bmod Max$$

+Ventaja: utiliza todas las cifras de la llave
-Inconveniente: es algo más costosa de calcular

- *Función Hash, suma de los códigos ASCII de los caracteres*

hash(llave) = $\left(\sum_{i=0}^{length-1}\left(ascii(llave_i)\right)\right) mod\ Max$

Donde Max es el tamaño de la tabla

+Ventajas: sencilla y eficiente
-Inconveniente: las llaves con los mismos caracteres (aunque estén en distinto orden) generan el mismo código de dispersión

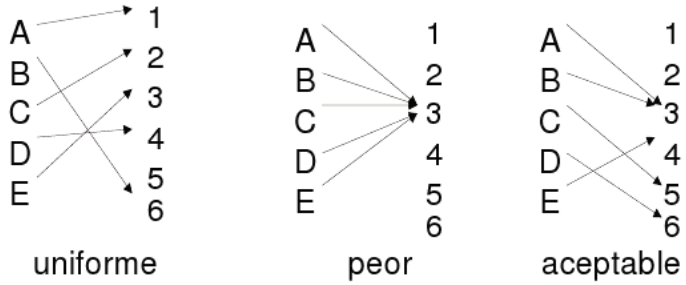
- *Función Hash, suma con pesos de los códigos ASCII de los caracteres*

hash(llave) = $\left(\sum_{i=0}^{length-1}\left(ascii\left(llave_i * peso^i\right)\right)\right) mod\ Max$

+Ventaja: el código depende de los caracteres que forman la llave y también del orden que ocupan dichos caracteres
-Inconveniente: más costosa de calcular

Colisiones

Una colisión de hash es una situación que se produce cuando dos entradas distintas a una función de hash producen la misma salida.
Es matemáticamente imposible que una función de hash carezca de colisiones, ya que el número potencial de posibles entradas es mayor que el número de salidas que puede producir un hash. Sin embargo, las colisiones se producen más frecuentemente en los malos algoritmos. En ciertas aplicaciones especializadas con un relativamente pequeño número de entradas que son conocidas de antemano es posible construir una función de hash perfecta, que se asegura que todas las entradas tengan una salida diferente. Pero en una función en la cual se puede introducir datos de longitud arbitraria y que devuelve un hash de tamaño fijo (como MD5), siempre habrá colisiones, debido a que un hash dado puede pertenecer a un infinito número de entradas.



Dispersión/Colisión: dos o más elementos tienen la misma posición:

Hash cerrado (Direccionamiento Abierto):

- Exploración lineal,

- Exploración cuadrática

Hash abierto (Direccionamiento Cerrado):

- Encadenamiento separado

Características de la Función HASH

- **Cálculo rápido**: Una buena función hash es simple, se puede calcular rápidamente, su principal ventaja es su velocidad. Si la función hash es lenta, esta velocidad será degradada. El propósito de una función hash es tomar una serie de valores y transformarlos en los valores de índice, de tal manera que los valores de clave están distribuidos al azar a través de todos los índices de la tabla hash. Las claves pueden ser totalmente aleatorias o no.

- **Claves aleatorias**: Llamamos a una función de hash "perfecta" si se mapea cada clave en una ubicación de tabla diferente. En la mayoría de los casos no existe esta situación, ya que la función de hash va a necesitar comprimir una gama más amplia de claves en un rango menor de números de índice que coincide con el rango del arreglo. Para lograr esto aplicamos el modulo a la función de hash para que de un valor dentro de ese rango:

  ndice =indice% arraySize

  Se trata de una sola operación matemática, y si las claves son realmente aleatorias, los índices resultantes serán aleatorios, y por lo tanto bien distribuidas.

- **Claves no Aleatorias**: En este caso los datos se distribuyen de manera no aleatoria. Imagínese que se usan como clave los números de autopartes. Estos números van a tener un formato particular, por ejemplo: 033-400-03-94-05-0-535 donde cada uno de los valores tiene una interpretación y un rango de valores posibles para cada uno. Por ejemplo, el 033 es un valor que representa una categoría cuyo valor posible es de 000 a 050. Estas claves no están distribuidas al azar.

- **Usar un número primo para el modulo**: Generalmente, la función de hash implica el uso del operador de módulo (mod o %) con el tamaño de la tabla. Además como veremos en la próxima sección, es importante que el tamaño de la tabla sea un número primo cuando utilizamos resolución de colisiones por *exploración lineal* o *cuadrática*. Sin embargo, si las claves no se distribuyen de manera aleatoria, es importante que el tamaño de la tabla se defina con un *número primo* independientemente del tipo de función de hashing que se utilice. Esto es así dado que si muchas claves comparten como divisor el tamaño del arreglo, puede tender a ubicar los elementos en la misma ubicación, causando la agrupación. Usando un número primo como tamaño de la tabla se elimina esta posibilidad.

Tipos de Hashing

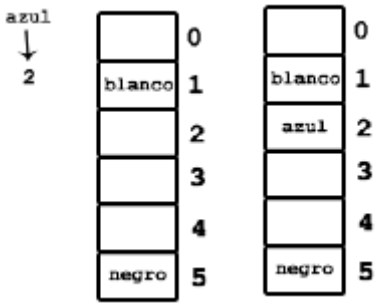
- ***Hashing Perfecto***: Existe una Función de Enumeración que asigna a cada valor del dominio una única posición de memoria. No posee colisiones.
- ***Hashing Puro***: La función de Hash puede asignar a dos valores distintos el mismo valor hash. Estos dos valores reciben el nombre de sinónimos. Las estructuras de hashing puros poseen

colisiones y en consecuencia se deberán establecer mecanismos para tratar los mismos. Podemos clasificarlos en estructuras cerradas y abiertas y dentro de las abiertas en estáticas y dinámicas:

- **Cerradas**: No utilizan un nuevo espacio en memoria.
- **Abiertas**: Utilizan espacio adicional.
  - ***Estática***: La estructura principal no crece.
  - ***Dinámica***: La estructura principal se expande a medida que aumenta la cantidad de elementos.

Tablas HASH (Operaciones Básicas)

Inserción: El proceso de inserción en una tabla hash es muy simple y sencillo. Sobre el elemento que se desea insertar se aplica la función de dispersión. El valor obtenido tras la aplicación de esta función será el índice de la tabla en el que se insertará el nuevo elemento.
Veamos este proceso con un ejemplo. Sobre la siguiente tabla hash se desea introducir un nuevo elemento, la cadena azul. Sobre este valor se aplica la función de dispersión, obteniendo el índice 2.

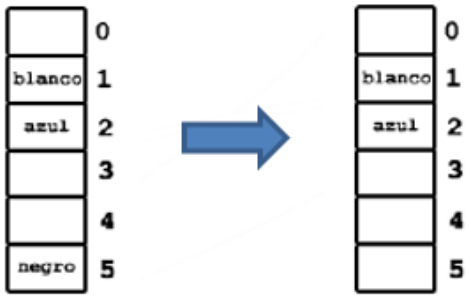


En el caso de que se produzca una ***colisión*** al tratar de insertar el nuevo elemento, el procedimiento será distinto en función del tipo de hash con el que se esté tratando, y se deberá tratar según la forma de resolución de colisiones implementada.

Búsqueda: Se implementa con la función de Hash en forma directa, a no ser que existan colisiones. En tal caso se deberá usar la técnica propuesta sobre los valores colisionados.

Borrado: El borrado en una tabla hash es muy sencillo y se realiza de forma muy eficiente.

Una vez indicada la clave del objeto a borrar, se procederá a eliminar el valor asociado a dicha clave de la tabla.
Esta operación *se realiza en tiempo constante, sin importar el tamaño de la tabla o el número de elementos que almacene* en ese momento la estructura de datos. Esto es así ya que al ser la tabla una estructura a la que se puede acceder directamente a través de las claves, no es necesario recorrer toda la estructura para localizar un elemento determinado.
Si sobre la tabla resultante de la inserción del elemento azul realizamos el borrado del elemento negro, la tabla resultante sería la siguiente:



Otras Operaciones: Una de las principales operaciones que se pueden realizar en las tablas hash es la ***redispersión***. Se suele realizar cuando el factor de carga (número de elementos / capacidad de la tabla) de la tabla supera cierto umbral.

La ***redispersión*** consiste en pasar todos los elementos de la tabla original a una nueva tabla de un tamaño mayor. De esta forma, se reduce el factor de carga de la tabla.

Resolución de colisiones

Hash cerrado (Direccionamiento Abierto):
- **Exploración lineal**
- **Exploración cuadrática**
- **Doble Hasheo**

Hash abierto (Direccionamiento Cerrado):
- **Encadenamiento separado**

Tablas HASH . Protección Activa

Es cuando se intentan evitar las colisiones diseñando buenas funciones hash

- *Función hash básica para claves enteras*

Buena opción: f(c) = c % B, siendo c la clave y B el tamaño de la tabla
Colisiones: n / B, siendo n el número de elementos y B el tamaño de la tabla

- *Función hash básica para claves string*

Hay que convertir la cadena a un valor numérico y luego aplicarle la función hash para claves enteras Ej. utilizando el código ASCII de cada carácter

Asignación de pesos para incrementar el rango
Una ponderación típica es 27^{2-i} siendo i la posición del carácter en la cadena

33	!	34	"	35	#	36	$	37	%	38	&	39	'		
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~		

Letra	Código ASCII	Código ponderado
P	80	$80 * 27^2 = 58320$
u	117	$117 * 27^1 = 3159$
e	101	$101 * 27^0 = 101$
r	114	
t	116	
a	97	
TOTAL:	**625**	**61580**

Asignación de pesos para incrementar el rango

Se puede optimizar la multiplicación utilizando como peso 32

Letra	Código ASCII	Código ponderado	Código ponderado (32)
P	80	$80 * 27^2 = 58320$	$80 * 32^5 = 2684354560$
u	117	$117 * 27^1 = 3159$	$117 * 32^4 = 122683392$
e	101	$101 * 27^0 = 101$	$101 * 32^3 = 3309568$
r	114		$114 * 32^2 = 116736$
t	116		$116 * 32^1 = 3712$
a	97		$97 * 32^0 = 97$
TOTAL:	**625**	**61580**	**2810468065**

Se puede minimizar el nº de multiplicaciones utilizando la Regla de Horner

$$p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

puede reescribirse como:

$$p(x) = ((((a_5x + a_4)x + a_3)x + a_2)x + a_1)x + a_0$$

$P*32^5 + u*32^4 + e*32^3 + r*32^2 + t*32^1 + a*32^0$

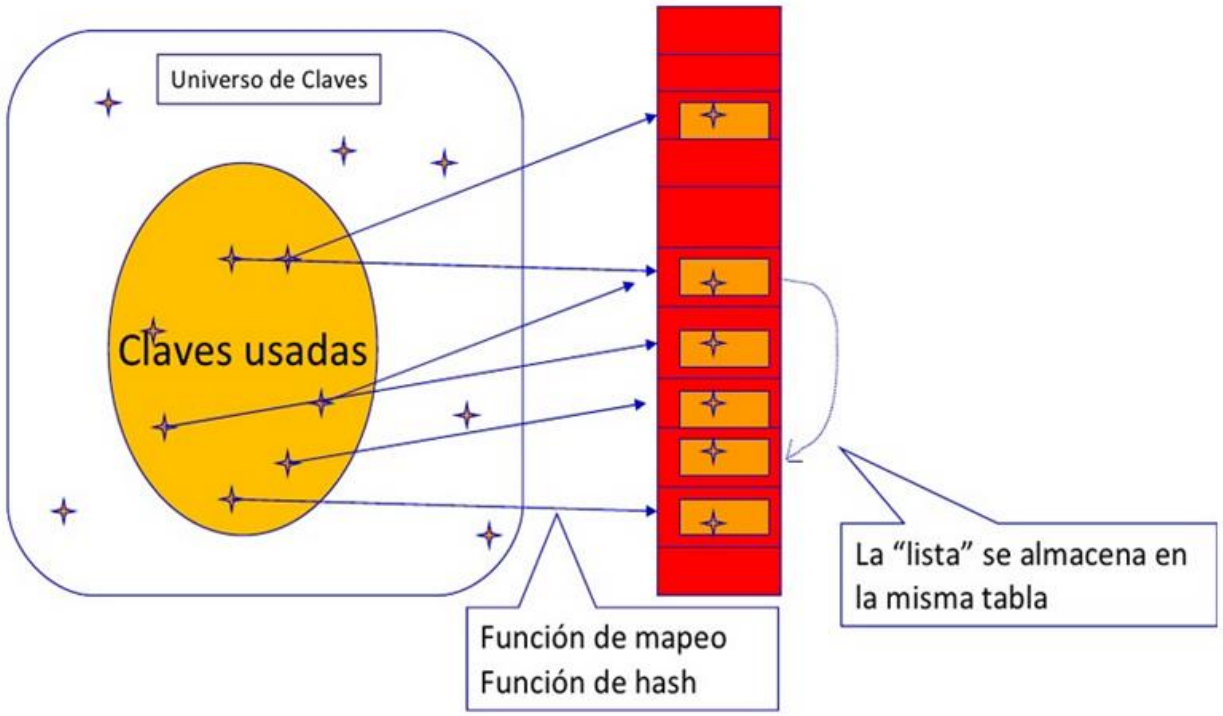
Tablas HASH . Protección Pasiva

Es cuando varios elementos necesitan compartir la misma posición dentro de la tabla

* Varios elementos comparten la misma posición de la tabla hash
  Ej.: cada posición de la tabla es a su vez una lista o un árbol
* Fac**tor de carga** (Load Factor) = n / B, siendo n el número de elementos y B el tamaño de la tabla
  Lo recomendable es que FC <= 1
* Cada posición sólo tiene cabida para un elemento
  Si se detecta una colisión, se buscan posiciones próximas
* Técnicas de búsqueda de posiciones próximas
  - Exploración lineal
  - Exploración cuadrática
  - Dispersión doble

Direccionamiento Abierto o Hash Cerrado

Cuando ocurre una colisión, podemos buscar de manera metódica una nueva posición vacía donde insertar el elemento en lugar de usar la posición generada por la función de hashing. Esta técnica es llamada direccionamiento abierto o hash cerrado.

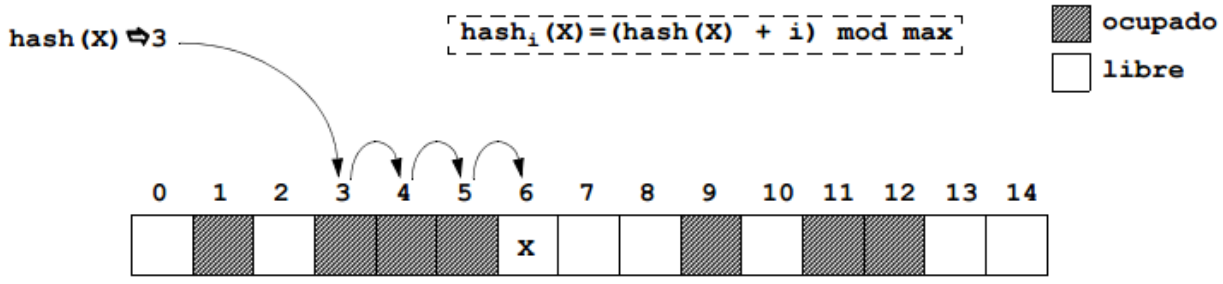


- *Exploración Lineal*

* Se modifica la función f(c) = c % B pasa a ser f(c) = (c + i) % B, siendo i = 0, 1, 2, 3, 4,…

La existencia de agrupamientos provoca problemas

- Cuando se busca un elemento que está en un agrupamiento hay que recorrer todos los elementos del agrupamiento hasta que se encuentra
- Lógicamente, cuando se encuentra una posición vacía no se sigue buscando (podría ser una complejidad O(n)) y se detiene el algoritmo



• Ocupación de la tabla: λ=celdasOcupadas/totalNumCeldas
• Número medio de intentos para encontrar celda libre=1/(1-λ)
  - si están ocupadas la mitad de las celdas (λ=0.5) en promedio se necesitan 2 intentos
  - Si la ocupación es mayor, el número de intentos crece mucho
• Si la función hash no es homogénea la eficiencia empeora
• Tienden a formarse bloques de celdas ocupadas que degradan la prestaciones

…pero, ¿y si se hubiera borrado algún elemento del agrupamiento?

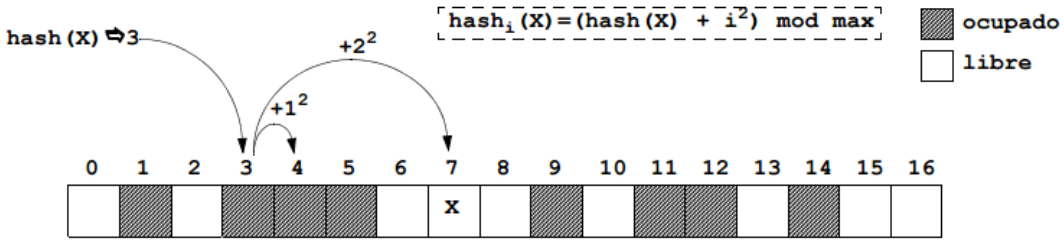
- *Exploración Lineal – Borrado prerezoso*

* No eliminar un elemento hasta que se introduce otro
* Simplemente, se marca
  Para insertar se considera que la posición está libre
  Para buscar se considera que la posición está ocupada

- *Exploración Cuadrática*

Para reducir el número de intentos se necesita un esquema de resolución de conflictos que evite la agrupación primaria. Si la función da como resultado la celda H y está ocupada, se consultan:

$$H + 1^2; H + 2^2; H + 3^2; \ldots ; H + i^2$$



• Evita en gran medida el agrupamiento de celdas ocupadas que ocurría con la exploración lineal
• Es preciso que la ocupación siempre sea inferior a la mitad
• El tamaño de la tabla debe ser un **número primo** para evitar que se formen ciclos en la función que excluyan celdas potencialmente vacías.

Teorema: Dada una tabla de dispersión que utiliza exploración cuadrática, si su tamaño es un número primo siempre podremos insertar un nuevo elemento en la tabla si su ocupación es estrictamente inferior a 0.5. Además, durante el proceso de inserción, no se visita la misma posición más de una vez

- *Dispersión Doble o Doble Hasheo*

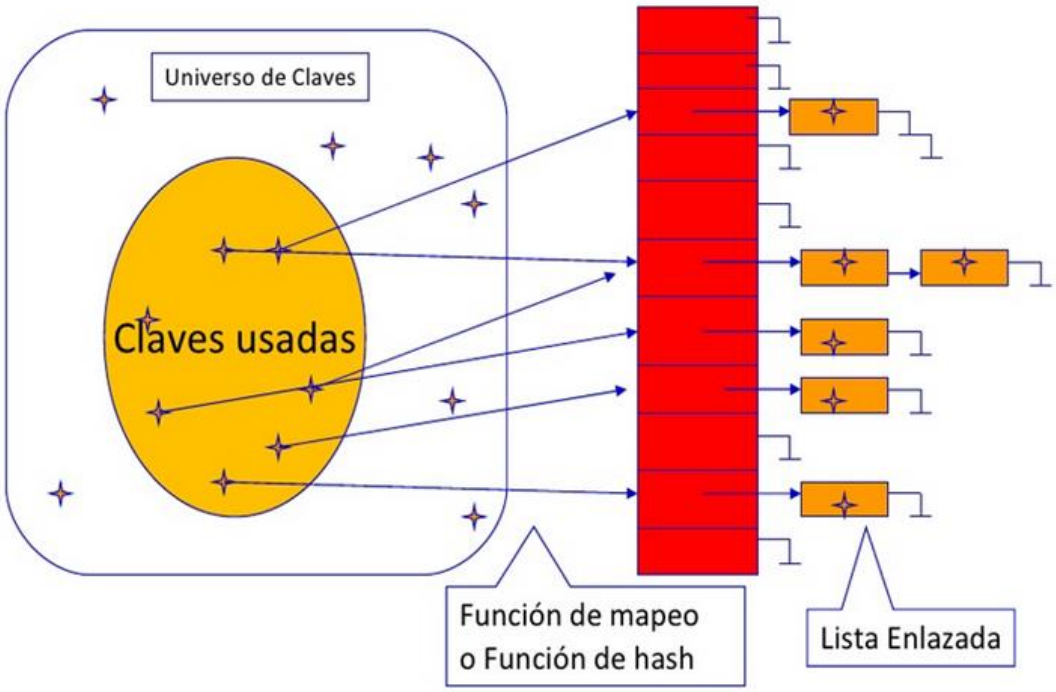
Permite eliminar la agrupación secundaria que se produce en ocasiones con la exploración cuadrática

$$\text{hash}_i(x) = (\text{hash}(x) + i \times \text{hash}_{alt}(x)) \bmod \text{max}$$

$\text{hash}_{alt}(x)$ nunca debe dar cero
- ej.: R-(x mod R), siendo R primo

Tablas Hash – Hash Abierto

Otra técnica que se puede aplicar cuando ocurre una colisión, consiste en crear una lista para cada posición del vector. De esta manera cuando se produce una colisión simplemente se agrega el elemento a la lista. Esta técnica es llamada direccionamiento cerrado o hash abierto.



Resolución por encadenamiento

El peor de los casos para la inserción es O(1). Para la búsqueda, el peor de los casos es proporcional al tamaño de la lista. La eliminación de un elemento x puede ser realizado en O(1) si la lista es doblemente encadenada.

- La inserción se hace como en una pila (al principio) ***con tiempo constante***
- La búsqueda es secuencial .
    - Lo peor que puede pasar es que nuestra función hash mapee concentrados todos los elementos (n) en un solo slot.
    - El caso promedio depende de cuan buena sea nuestra función hash. *(suponiendo que cualquier elemento n tenga la misma probabilidad de ser mapeado a cualquiera de los m slots de T, independientemente de los otros)*..

Para j=0, 1, … , m-1 y sea n_j el largo de la lista T[j], entonces

N= $n_0 + n_1 + n_2 + \ldots + n_j + \ldots + n_{m-1}$

el valor medio es $E[n_j] = \alpha = \frac{n}{m}$

$$\sum_{i=0}^{m-1} p_i = 1 \xrightarrow[\text{Uniform Hashing}]{\text{Simple}} p \underbrace{\sum_{i=0}^{m-1} 1}_{m} = 1 \Rightarrow pm = 1 \Rightarrow p = \frac{1}{m}$$

Teorema 1
*En una tabla hash en la cual las colisiones son resueltas con encadenamiento, una búsqueda **sin éxito** toma un tiempo Θ(1 + α), en promedio, bajo la suposición de hashing uniforme simple.*

Teorema 2
*En una tabla hash en la cual las colisiones son resueltas con encadenamiento, una búsqueda **exitosa** toma un tiempo Θ(1 + α), en promedio, bajo la suposición de hashing uniforme simple.*

• El análisis anterior significa que si el numero de slots, en una tabla hash, es proporcional al numero de elementos en la tabla, se tiene n = O(m) y de esta forma, α = n/m = O(m)/m = O(1).
• Por lo tanto, la búsqueda toma tiempo constante en promedio.
• Como la inserción toma O(1) en el peor de los casos, y el borrado toma O(1) en el peor de los casos cuando las listas están doblemente encadenadas, todas las operaciones pueden tomar tiempo constante en promedio.

Queremos una función que satisfaga "**Simple Uniform Hashing**"
Pero eso no es posible si no conocemos las entradas, ya que puede ser sesgada.

- Método de la división
  h(k)= K mod m $K \in \mathbb{N} + el\ 0$
  Tener una potencia de 2 es deseable porque nos simplifica el calculo pero

  Necesitamos que m sea un numero primo alejado de una potencia de 2
- Método de Multiplicación

  h(k)= ⌊m(KA mod 1)⌋ 0<A<1 ⌈ ⌉ Trunca al entero superior
  KA mod 1=KA – ⌊KA⌋ ⌊ ⌋ Trunca al entero inferior

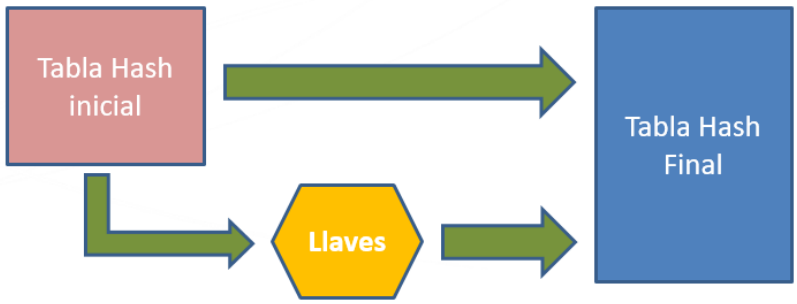
El valor de m no es critico y m es normalmente una potencia de 2

Tablas HASH . Redispersión o Rehash

¿Qué sucede cuando no se puede insertar un elemento debido a que todas las posiciones colisionan?

* Aumentar el tamaño en función del FC
    Tablas hash abiertas à rendimiento decrece si FC > 1
    Tablas hash cerradas à se paraliza si FC > 0,5
* Se busca un nuevo valor B
    El primo inmediatamente superior al tamaño doble del original
    Se recorren los elementos y se añaden a la nueva tabla

Aumentar el tamaño de la tabla y volver a asociar los elementos.



Tablas HASH . Hashing Dinámico

• La función de hash se va modificando de acuerdo al número de elementos y al comportamiento de la tabla.
• Hashing extensible (**Extendible hashing**).
    – Uso de un directorio con M registros.
    – Concatenación de múltiples tablas hash.